# New Optimizer I

Alexander Ludwig

Deep Learning Research Kitchen (ML-4501 / 3 ECTS)
June 12, 2024

# Structure

New but famous optimizers
Evidence Lower Bound & Entropy Coding

# AdamW: decoupling weight decay from your adaptive optimizer

## more state of the art research from Freiburg

- Given a loss $l(\theta) = \ldots + \lambda||\theta||^2$
- only for vanilla SGD this is equivalent to **weight decay**
  $\theta_{t+1} = \theta_t - \eta(\cdots + \lambda\theta_t)$

  $\Longrightarrow$ first update $\theta_{t+1} = -\eta\lambda\theta_t$ then *continue*
- AdamW is one of the best practice optimizers (at least firmly ingrained in NanoGPT)


DECOUPLED WEIGHT DECAY REGUL

**Ilya Loshchilov & Frank Hutter**
University of Freiburg
Freiburg, Germany,
{ilya,fh}@cs.uni-freiburg.de

# But why should we stick to first order methods?

## Reminder: Antionio's Lecture and Newton's Method

"With $H = L \cdot I_{dxd}$ the theory is perfect"
"the optimal $\eta$, yielding the maximum decrease is $\eta = 1/L$ "

**Hessian Matrix**, $\theta \in \mathbb{R}^d$

$$H = \begin{bmatrix} \frac{\partial^2 L}{\partial \theta_1^2} & \frac{\partial^2 L}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 L}{\partial \theta_1 \partial \theta_n} \\ \frac{\partial^2 L}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 L}{\partial \theta_2^2} & \cdots & \frac{\partial^2 L}{\partial \theta_2 \partial \theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 L}{\partial \theta_n \partial \theta_1} & \frac{\partial^2 L}{\partial \theta_n \partial \theta_2} & \cdots & \frac{\partial^2 L}{\partial \theta_n^2} \end{bmatrix} \in \mathbb{R}^{d \times d}$$

**Newton's Method**: $\theta_{t+1} = \theta_t - H^{-1} g$
"quadratic convergence for convex problems"

# Gauss-Newton-Bartlett Estimator [1]

## making the Hessian diagonal and linearize a deep neural network

Reminder the multivariate Chain Rule $L : L(f_1(x), f_2(x), \ldots, f_c(x))$:

$$\frac{\partial L}{\partial x_i}(f(x)) = \sum_{j=1}^{c} \frac{\partial L}{\partial f_j} \frac{\partial f_j}{\partial x_i}$$

$$H_{ij} = \frac{\partial^2 L}{\partial \theta_i \partial \theta_j} = \frac{\partial}{\partial \theta_i} \left( \sum_j \frac{\partial L}{\partial f_j} \frac{\partial f_j}{\partial \theta_i} \right) = \left( \sum_j \frac{\partial}{\partial \theta_i} \left( \frac{\partial L}{\partial f_j} \right) \frac{\partial f_j}{\partial \theta_i} + \frac{\partial L}{\partial f_j} \frac{\partial}{\partial \theta_i} \left( \frac{\partial f_j}{\partial \theta_i} \right) \right)$$

$$= \left( \sum_j \sum_k \frac{\partial^2 L}{\partial f_j \partial f_k} \frac{\partial f_j}{\partial \theta_i} \frac{\partial f_k}{\partial \theta_j} \right) + \sum_j \left( \frac{\partial L}{\partial f_j} \underbrace{\frac{\partial}{\partial \theta_i} \left( \frac{\partial f_j}{\partial \theta_i} \right)}_{=0 \text{ for linear} f} \right) = J_\theta^T \underbrace{H_f}_{\in \mathbb{R}^{c \times c}} J_\theta + smol$$

[1] https://youtu.be/416NjW3QfwA?feature=shared (great Lecture about Second Order Optimizer from the Numerics of ML lecture by Lukas Tatzel)

# GNB 2

Since $L = H(p_{\text{data}}, p_{\text{model}}) = E_{x \sim p_{data}}[-\log(p_{model}(x))]$ one can rewrite
$H_f = E_{\hat{y} \sim p_{model}}[\frac{\partial^2 L}{\partial^2 f}] = E_{\hat{y} \sim p_{model}}[\frac{\partial L}{\partial f}\frac{\partial L}{\partial f}^T]$ ($H_f$ does no independent on $\hat{y}$)
$\rightarrow G = E[J_\theta \frac{\partial L}{\partial f}\frac{\partial L}{\partial f}^T J_\theta^T] = E_{\hat{y} \sim p_{model}}[\nabla_\theta L \odot \nabla_\theta L]$

- splitting up second derivatives in expectation is called Bartlett's second identity
- sample $\hat{y} \sim p_{model}$ and calculate gradient w.r.t. $L$

# Sophia

## Second-order Clipped Stochastic Optimization

- accounts for curvature information during optimization
- clips gradient at 1
- "The authors suspect the GNB estimator has a smaller variance than the Hutchinson's estimator, [...]."
- public implementation only with GNB available

**Algorithm 3** Sophia

**Input:** $\theta_1$, learning rate $\{\eta_t\}_{t=1}^{T}$, hyperparameters $\lambda, \gamma, \beta_1, \beta_2, \epsilon$, and estimator choice Estimator $\in$ {Hutchinson, Gauss-Newton-Bartlett}
Set $m_0 = 0$, $v_0 = 0$, $h_{1-k} = 0$
**for** $t = 1$ **to** $T$ **do**
  Compute minibach loss $L_t(\theta_t)$.
  Compute $g_t = \nabla L_t(\theta_t)$.
  $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  **if** $t \bmod k = 1$ **then**
    Compute $\hat{h}_t = \text{Estimator}(\theta_t)$.
    $h_t = \beta_2 h_{t-k} + (1 - \beta_2)\hat{h}_t$
  **else**
    $h_t = h_{t-1}$
  $\theta_t = \theta_t - \eta_t \lambda \theta_t$ (weight decay)
  $\theta_{t+1} = \theta_t - \eta_t \cdot \text{clip}(m_t / \max\{\gamma \cdot h_t, \epsilon\}, 1)$

Claude Shannon

# Defining 2x faster

$O_2$ is $k$-times faster than $O_1$ if

$$\exists H_2, \quad \min_{H_1} \mathrm{Eval}(O_1, T, H_1) \geq \mathrm{Eval}(O_2, T/k, H_2)$$

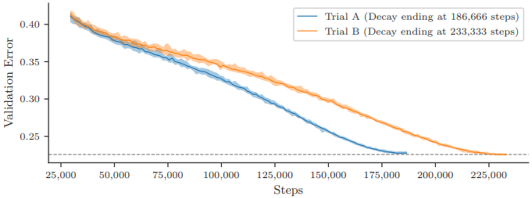



Figure 4: **For a fair comparison, training curves need to be tuned for the same criterion.** Two AdamW training runs (—, —) for ResNet-50 on ImageNet using hyperparameters tuned within the same search space, but using different step budgets. Since the cosine learning rate decay schedule stretches with a longer step budget, we see "slower" training caused by the larger step budget (—). For each of the hyperparameter settings, we ran 20 different random seeds to create min/max error bounds around a median trajectory (■, ■). The dashed gray line (- -) denotes the best median validation error achieved by both training runs. See Appendix A.4.3 for experimental details.

# Lion: EvoLved Sign Momentum, an attempt to gödelize optimizers

## Why not breed your dream optimizer via evolution

- regularized evolution searches through the symbolic representations
- inputs are weights, lr and gradient; output can be weights update + extra stuff
- performance is measured on ViT on 10% of Imagenet (20 mins TPU) and very tiny on LM1B
- limited vocabulary and constants are modified/ newly sampled a la $2^a$ for $a \sim \mathcal{N}(0, 1)$
- prune search space by checking for functional equivalence and redundant statements

Program 8: Raw program of Lion before removing redundent statements.

```
def train(w, g, m, v, lr):
  g = clip(g, lr)
  m = clip(m, lr)
  v845 = sqrt(0.6270633339881897)
  v968 = sign(v)
  v968 = v - v
  g = arcsin(g)
  m = interp(g, v, 0.8999999761581421)
  v1 = m * m
  v = interp(g, m, 1.109133005142212)
  v845 = tanh(v845)
  lr = lr * 0.0002171761734643951
  update = m * lr
  v1 = sqrt(v1)
  update = update / v1
  wd = lr * 0.4601978361606598
  v1 = square(v1)
  wd = wd * w
  m = cosh(update)
  lr = tan(1.4572199583053589)
  update = update + wd
  lr = cos(v845)
  return update, m, v
```

# Lion 2

Optimizing NNs is already searching over TMs, where will this lead to when we now start searching optimizers?

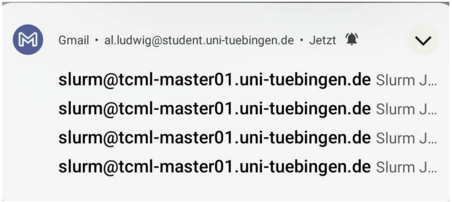
- decoupled weight decay is added manually (gray lines)
- `interp(a,b,`$\lambda$`)` = $(1-\lambda)a + \lambda b$
- uniform update direction across **all** dimensions of $\theta$
- authors argue that sign operation adds noise to gradient update and regularizes
- smaller memory footprint than AdamW (only momentum stored)
- almost the same as signSGD (moment variant)

Program 1: Discovered optimizer Lion. $\beta_1 = 0.9$ and $\beta_2 = 0.99$ by default are derived from Program 4. It only tracks momentum and uses the sign operation to compute the update. The two gray lines compute the standard decoupled weight decay, where $\lambda$ is the strength.

```
def train(weight, gradient, momentum, lr):
  update = interp(gradient, momentum, β1)
  update = sign(update)
  momentum = interp(gradient, momentum, β2)
  weight_decay = weight * λ
  update = update + weight_decay
  update = update * lr
  return update, momentum
```

Cleaned up optimizer

# Experiments




```
[rank1]: RuntimeError: Found NVIDIA GeForce GTX 1080 Ti which is too old to be supported by the triton GPU compiler, w
hich is used as the backend. Triton only supports devices of CUDA Capability >= 7.0, but your device is of CUDA capabi
lity 6.1
```

sbatch: error: QOSMaxWallDurationPerJobLimit

# *#SBATCH --partition=day*

### 2.2.2 Partitions (Queues):

4 partitions with different time limits are provided:

| Partition name | TimelLimit | Always accessible nodes | Additional nodes accessible if unused |
|---|---|---|---|
| **test (default)** | 15 min | 3 | 37 |
| **day** | 1 day | 10 | 25 |
| **week** | 7 days | 17 | 10 |
| **month** | 30 days | 10 | 0 |

https://csweb.cs.uni-tuebingen.de/webprojects/TCML/

# Setup

- All Experiments are run on the tcml cluster on a single computer node with 4x GeForce GTX 1080 Ti and Intel XEON CPU E5-2650 v4 (24 cores)
- fork of Sophia repository which itself is a fork of NanoGPT repository by Andrej Karpathy
- 10K update steps during training on data from OpenWebText 2
- cross entropy loss
- *tiny* transformer (30M parameters, 6 Layers, 6 attention heads, 384 embedding dimension, context length of 1024 tokens)
- *gradient accumulation* was used to simulate a higher batch size (120 most of the time)

# Hyperparameters

- Sophia paper reports *optimal* hyperparameters for tiny model on 50K (40K more) steps for different optimizers
- increase maximum learning rate until divergence
-
-
-

# Summary: What is the empirical bit rate for data compression in MRI?

# negative ELBOW